Homework 2

Due by **11:59pm, Thursday, September 20, 2018**.

Make sure you follow all the homework policies (../../policies/hw-policy.html).

All submissions should be done via Autolab (../../autolab.html).

Sample Problem

The Problem

This problem is just to get you thinking about asymptotic analysis and input sizes.

An integer $n \geq 2$ is a prime, if the only divisors it has is 1 and n. Consider the following algorithm to check if the given number n is prime or not:

For every integer $2 \leq i \leq \sqrt{n}$, check if i divides n. If so declare n to be *not* a prime. If no such i exists, declare n to be a prime.

What is the function $f(n)$ such that the algorithm above has running time $\Theta(f(n))$? Is this a polynomial running time-- justify your answer. (A tangential question: Why is the algorithm correct?)

Click here for the Solution

Submission

You will **NOT** submit this question. This is for you to get into thinking more about asymptotic analysis.

Question 1 (Programming Assignment) [30 points]

</> Note

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

The Problem

In this problem we will consider the extension of the stable matching problem that more closely resembles the resident matching program that NRMP (http://www.nrmp.org/) administers.

We are given m hospitals and n medical students. Each hospital has a ranking of *all* the students in order of preference, and each student has a ranking of *all* the hospitals in order of preference. Unlike the stable matching problem, a hospital *can* have more than one open slots (but a student can be assigned at most one hospital). We will assume that there are more students than there are slots available in all the m hospitals put together (they can be equal). The goal is to output a **stable** assignment of students to hospitals. Note that since there are more students than hospitals, some students may be not assigned to any hospital. Also no hospital is assigned more students than the number of openings it has.

An assignment of students to hospital is stable if neither of the following situation arises:

1. First type of instability: There are students s and s', and a hospital h, so that:
    - s is assigned to h, and
    - s' is assigned to no hospital, and
    - h prefers s' to s.
2. Second type of instability: There are students s and s', and hospitals h and h', so that:
    - s is assigned to h, and
    - s' is assigned to h', and
    - h prefers s' to s, and
    - s' prefers h to h'.

Input

The input is an instance of the national resident problem in a text file of the following format:

```
m                                                  <- Number of hospitals
n                                                  <- Number of students
s0^1  s1^1  s2^1  s3^1 ...  sn^1                    <- Preference of the 1st hospital (mos
s0^2  s1^2  s2^2  s3^2 ...  sn^2                    <- Preference of the 2nd hospital (mos
s0^3  s1^3  s2^3  s3^3 ...  sn^3                    <- Preference of the 3rd hospital (mos
s0^4  s1^4  s2^4  s3^4 ...  sn^4                    <- Preference of the 4th hospital (mos
.
.
.
s0^m  s1^m  s2^m  s3^m ...  sn^m                     <- Preference of the mth hospital (mos
h1^1  h2^1  h3^1 ...  hm^1                       <- Preference of the 1st student (most pre
h1^2  h2^2  h3^2 ...  hm^2                       <- Preference of the 2nd student (most pre
h1^3  h2^3  h3^3 ...  hm^3                       <- Preference of the 3rd student (most pre
h1^4  h2^4  h3^4 ...  hm^4                       <- Preference of the 4th student (most pre

.
.
.
h1^n  h2^n  h3^n ...  hm^n                       <- Preference of the nth student (most pre
```

For example

```
3                                          <- Number of hospitals
5                                          <- Number of students
1   2   3   5   1   4                      <- Preference of the 1st hospital (most prefe
1   5   1   2   4   3                      <- Preference of the 2nd hospital (most prefe
2   5   2   3   1   4                      <- Preference of the 3rd hospital (most prefe
2   1   3                                  <- Preference of the 1st student (most prefer
3   2   1                                  <- Preference of the 2nd student (most prefer
3   1   2                                  <- Preference of the 3rd student (most prefer
1   2   3                                  <- Preference of the 4th student (most prefer
1   2   3                                  <- Preference of the 5th student (most prefer
```

Note

A hospital can have more than one available slot which is stored in the first index of its preference list.

Output

The output is an instance of stable matchings for the input in a text file of the following format:

```
(h1, s1)            <- Pairing of the form (h,s)
(h2, s2)            <- Pairing of the form (h,s)
(h3, s3)            <- Pairing of the form (h,s)
.
.
.
(hn, sn)            <- Pairing of the form (h,s)
```

For example:

```
(1, 5)                          <- Pairing of the form (h,s)
(2, 1)
(3, 2)
(3, 3)
```

Note

We note that we will assume that a student that is not assigned to any hospital is not part of the output. **More importantly**, please note that there is more than one possible stable assignments possible for any given input. I.e. even if your algorithm is correct, its output might not match the given sample output. However, as long as your output is a stable assignment, you should be fine.

Hint

The best possible algorithm for this problems that we are aware of runs in time $O(m \cdot n)$.

! Note

Both the input and output parsers in each of the three languages are already written for you. Note that you have to work with the input data structures provided (which will come pre-loaded with the data from an input file).

! Addition is the only change you should make

Irrespective of what language you use, you will have to submit just one file. That file will come pre-populated with some stuff in it. You should **not** change any of those things because if you do you might break what the grader expects and end up with a zero on the question. You should of course add stuff to it (including helper functions and data structures as you see fit).

Java | Python | C++

Download Java Skeleton Code (HW2Java.zip)

Directory Structure

```
├── src
│   ├── ub
│       ├── cse
│           ├── algo
│               ├── Driver.java
│               ├── HW2Utility.java
│               ├── Match.java
│               ├── PreferenceLists.java
│               ├── Solution.java
├── testcases/
│   ├── input1.txt
│   ├── input2.txt
│   ├── input5.txt
├── outputs/
│   ├── output1.txt
│   ├── output2.txt
│   └── output5.txt
```

You are given five coding files: `Driver.java`, `HW2Utility.java`, `Match.java`, `PreferenceLists.java` and `Solution.java`. `Driver.java` takes the input file, parses it and creates an instance of the class and prints result in output file. You only need to update the `Solution.java` file. You may write your own helper methods and data structures in it.

The testcases folder has 3 input files and their corresponding output files for your reference. We will use these three input files (and seven others) in our autograding.

Method you need to write:

```
5.
/**
 * This method must be filled in by you. You may add other
 * but they must remain within the HW2_Student_Solution c
 * @return Your set of stable matches. Order does not mat
 */
public ArrayList<Match>> getMatches() {

  return new ArrayList<Match>>();
}
```

Note: The output for this function must output an `Arraylist` of `Match`

Do not include students that did not get matched with a hospital in the output.

The `Solution` class has 4 instance variables.

- `_nHospital` which is of type int and stores number of hospitals.
- `_nStudent` which is of type int and stores number of students.

- `_hospitalList` which is of type `HashMap<Integer, ArrayList<Integer>>` and stores the preference lists of hospitals. Please note that the front of the `ArrayList<Integer>` (index 0) denotes the number of available slots.
- `_studentList` which is of type `HashMap<Integer, ArrayList<Integer>>` and stores the preference lists of students. Please note that the front of the `ArrayList<Integer>` (index 0) denotes the most preferred hospital.

The Other files

`Match.java` defines the `Match` class. This should be fairly intuitive. Below is the entire content of the class:

```
public class Match implements Comparable{
    public Integer student;
    public Integer hospital;

    Match(Integer student, Integer hospital) {
        this.student = student;
        this.hospital = hospital;
    }

    @Override
    public boolean equals(Object obj) {
        Match compare = (Match) obj;
        return (student.equals(compare.student)) && (hosp
    }
    @Override
    public String toString() {
        return "(" + student + ", " + hospital + ")";
    }
    @Override
    public int compareTo(Match other) {
        return this.hospital.compareTo(other.hospital);
    }
}
```

The file `HW2Utility.java` handles some of the background stuff: e.g. `readFile` reads in the file passed as a command line argument to `Driver.java` and populates the preference lists within the `HW2Utility` class.

Compiling and executing from command line:

Assuming you're in the same directory level as `src` . Run `javac src/ub/cse/algo/*.java` to compile.

To execute your code on `input1.txt` , run `java -cp "src" ub.cse.algo.Driver testcases/input1.txt` .

Submission

You only need to submit `Solution.java` to Autolab.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading).

Question 2 (Home-wrecker) [45 points]

The Problem

The stable marriage problem does not handle divorces. This is because we assume everyone is interested in everyone else of the opposite sex and we assume that the preferences *do not change*.

In this problem, we will see the effect of changes in preferences in the outcome of the Gale-Shapley algorithm (for this problem you can assume the version of the Gale-Shapley algorithm that we did in class where the women do all the proposing).

Given an instance of the stable marriage problem (i.e. set of men M and the set of women W along with their preference lists: L_m and L_w for every $m \in M$ and $w \in W$ respectively), call a man $m \in M$ a `home-wrecker` if the following property holds. There exists an L'_m such that if m changes his preference list to L'_m (from L_m) then the Gale-Shapley algorithm matches everyone to someone else. In other words, let S_{orig} be the stable marriage output by the Gale-Shapley algorithm for the original input and S_{new} be the stable marriage output by the Gale-Shapley algorithm for the new instance of the problem where m's preference list is replaced by L'_m (but everyone else has the same preference list as before). Then

$$S_{\text{orig}} \cap S_{\text{new}} = \emptyset.$$

Here are the two parts:

- Part `(a)`:
  You will first solve a simpler version of the problem. For every integer $n \geq 2$ argue the following: There exists an instance of the stable marriage problem with n men and n women such that there is a man m such that is he changes his preference list from L_m to L'_m then we have

  $$S_{\text{orig}} \neq S_{\text{new}}.$$

- Part `(b)`: For every integer $n \geq 2$ argue the following: There exists an instance of the stable marriage problem with n men and n women such that there is a man who is a `home-wrecker`.

⊘ Common Mistake

A common mistake for this problem is to present an instance that solves part `(a)` but not part `(b)`.

Note

To get **full credit**, you should be able to present an instance for *every* $n \geq 2$ in full detail. In other words, you have to present a "family" of examples (i.e. for each $n \geq 1$, you have to present the original $2n$ preference lists).

Hint

> Note that you have the freedom to decide how the Gale-Shapley algorithm chooses a free woman at any iteration of the while loop in case there is more than one choice.

Walkthrough video

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible.

> **!** PDF only please
>
> Autolab might not be able to display files in formats other than PDF (e.g. Word cannot be displayed). **If Autolab cannot display your file, then you will get a zero (0) on the entire question. Note that Autolab will NOT give an error message if you submit non-PDF file, so it is YOUR responsibility to make sure you submit in the correct format.** Also the file size has to be **at most 3MB**.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to part `(a)` of this problem:

1. `Proof idea : 10` points for outlining your instances for every $n \geq 2$ as well as specifying what S_{orig} and S_{new} are for each instance.

and here is the high level grading rubric for part `(b)`:

1. `Proof idea`: `17` points for outlining your instances for every $n \geq 2$.
2. `Proof details`: `18` points for a detailed description of your instance for each $n \geq 2$. (This is worth 13 points.) You also have to argue why your family of instance proves what is needed to be proven-- in other words, you need to argue why the Gale Shapley algorithm outputs your claimed S_{old} and S_{new} (note that this implies you also need to specify S_{orig} and S_{new}). This can be at the level of proof idea: proof details for this part are not needed. (This part is worth 5 points.) However, note that if the grader cannot understand why your construction works immediately, then you might (and most probably will) lose points. So it is in your best interest to make sure that is enough intuition given on why your construction works.
3. `Note`: If you only do part `(b)`, then your score on `Proof Idea` for this part will be counted as your score for part `(a)` (scaled to `10` points of course).

! Note

If you do not have separated out proof idea and proof details for part `(b)`, you will get a zero(0) irrespective of the technical correctness of your solution..

Templates

Download LaTeX template. (latex-template.zip)

Download Microsoft Word template. (word-template.zip)

! Note

Your must explicitly list your sources and collaborators when you upload your submission to Autolab. Note that there are only five allowed sources (../../policies/hw-policy.html). If you have used a source that is not allowed, please do **not** submit your homework. If you did not consult any source or did not collaborate with anyone just say `None`.

Questions 3 (Big G is in town) [25 points]

The Problem

The `Big G` company in the bay area decides it has not been doing enough to hire CSE grads from UB so it decides to do an exclusive recruitment drive for UB students. The `Big G` decides to fly over n CSE majors from UB to the bay area during December for on-site interview on a single day. The company sets up m slots in the day and arranges for n `Big G` engineers to interview the n UB CSE majors. (You can and should assume that $m > n$.) The fabulous scheduling algorithms at `Big G` 's offices draw up a schedule for each of the n majors so that the following conditions are satisfied:

- Each CSE major talks with every `Big G` engineer exactly once;
- No two CSE majors meet the same `Big G` engineer in the same time slot; and
- No two `Big G` engineers meet the same CSE major in the same time slot.

In between the schedule being fixed and the CSE majors being flown over, the `Big G` engineers were very impressed with the CVs of the CSE majors (including, ahem, their performance in CSE 331) and decide that `Big G` should hire all of the n UB CSE majors. They decide as a group that it would make sense to assign each CSE major S to a `Big G` engineer E in such a way that after S meets E during her/his scheduled slot, all of S's and E's subsequent meetings are canceled. Given that this is December, the `Big G` engineers figure that taking the CSE majors out to the nice farmer market at the ferry building in San Francisco during a sunny December day would be a good way to entice the CSE majors to the bay area.

In other words, the goal for each engineer E and the major S who gets assigned to her/him, is to **truncate** both of their schedules after their meeting and cancel all subsequent meeting, so that no major gets **stood-up**. A major S is stood-up if when S arrives to meet with E on her/his truncated schedule and E has already left for the day with some other major S'.

Your goal in this problem is to design an algorithm that always finds a valid truncation of the original schedules so that no CSE major gets stood-up.

To help you get a grasp of the problem, consider the following example for $n = 2$ and $m = 4$. Let the majors be S_1 and S_2 and the `Big G` engineers be E_1 and E_2. Suppose S_1 and S_2's original schedules are as follows:

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2	free
S_2	free	E_1	free	E_2

In the above schedules "free" means that the student is not meeting any engineer.

In this case the (only) valid truncation is for S_1 to get assigned to E_2 in the third slot and for S_2 to get assigned to E_1 in the second slot. So the truncated schedule will look like this:

CSE Major	Slot 1	Slot 2	Slot 3	Slot 4
S_1	E_1	free	E_2 (truncate here)	
S_2	free	E_1 (truncate here)		

Here are the two parts:

- Part `(a)` : In this part, we will see two algorithms that *do not work*. Your job will be, for each algorithm, to give an example input where the algorithm does not work (and briefly explain why that is the case).
  - `Engineer-centric Algorithm` : the first algorithm considers all engineers E and say E's last meeting is with CSE major S, then both E and S's schedules are truncated after their meeting.
  - `CSE major-centric Algorithm` : the second algorithm considers all CSE majors S' and say S's last meeting is with engineer E', then both E' and S''s schedules are truncated after their meeting.
- Part `(b)` : Design an *efficient* algorithm that always finds a valid truncation of the original schedules so that no CSE major gets stood-up.

Hint

In real life, you will almost never come across a problem whose description will match exactly with one you will see in this course. More often, you will come across problems that you have seen earlier *but* are stated in a way that don't look like the version you have seen earlier. *One way* to solve this problem would be to to simulate that situation. In algorithms-speak, you can to reduce (../../../support/reduction/index.html) the problem here to one that you have seen already.

⊘ Common Mistake

A common mistake for this part is to present an algorithm that does not take into account both the engineers' and CSE majors' schedules together.

Walkthrough video

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible.

! PDF only please

Autolab might not be able to display files in formats other than PDF (e.g. Word cannot be displayed). **If Autolab cannot display your file, then you will get a zero (0) on the entire question. Note that Autolab will NOT give an error message if you submit non-PDF file, so it is YOUR responsibility to make sure you submit in the correct format.** Also the file size has to be **at most 3MB**.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to part `(a)` of this problem:

1. For both algorithms, `Proof idea : 5` points *each*. In both cases you need to supply a counter-example *and*argue why your example shows that the corresponding algorithm is not correct.

and here is the high level grading rubric for part `(b)` :

1. `Algorithm idea : 4` points for explaining the idea behind your algorithm or a reduction to a familiar problem.
2. `Algorithm details : 3` points for providing the specific details of the algorithm or the reduction.
3. `Proof idea : 4` points for a proof idea that you can always achieve a valid truncation.
4. `Proof details : 4` points for providing the proof details.

! Note

If you do not have separated out algorithm idea, algorithms details, proof idea, proof details for part `(b)` , you will get a zero(0) irrespective of the technical correctness of your solution..

Templates

Download LaTeX template. (latex-template.zip)

Download Microsoft Word template. (word-template.zip)

! Note

Your must explicitly list your sources and collaborators when you upload your submission to Autolab. Note that there are only five allowed sources (../../policies/hw-policy.html). If you have used a source that is not allowed, please do **not** submit your homework. If you did not consult any source or did not collaborate with anyone just say `None` .
